CC (After mid) 1st

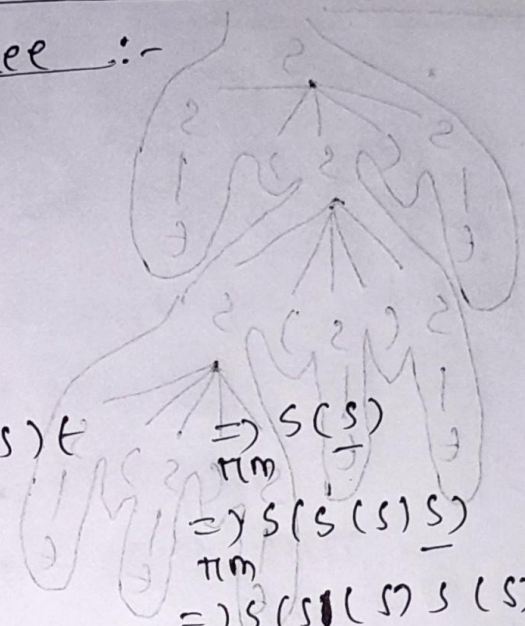
03.05.23

construct a parse tree :-

$S \rightarrow S(S)S$

$|\epsilon$

string :- (()())



So,

$S \underset{rm}{\Rightarrow} S(S)\underline{S} \underset{rm}{\Rightarrow} S(S)\epsilon \underset{rm}{\Rightarrow} S(\underline{S})$

$\underset{rm}{\Rightarrow} S(S(S)\underline{S})$

$\underset{rm}{\Rightarrow} S(S(S)S(S)\underline{S})$

$\underset{rm}{\Rightarrow} S(S(S)S(S)\epsilon))$

$\underset{rm}{\Rightarrow} S(S(S)S(\underline{S}))$

$\underset{rm}{\Rightarrow} S(S(S)S(\epsilon))$

$\underset{rm}{\Rightarrow} S(S(S)\underline{S}())$

$\underset{rm}{\Rightarrow} S(S(S)\epsilon())$

$\underset{rm}{\Rightarrow} S(S(\underline{S})())$

$\underset{rm}{\Rightarrow} S(S(\epsilon)())$

$\underset{rm}{\Rightarrow} S(\underline{S}()())$

$\underset{rm}{\Rightarrow} S(\epsilon()())$

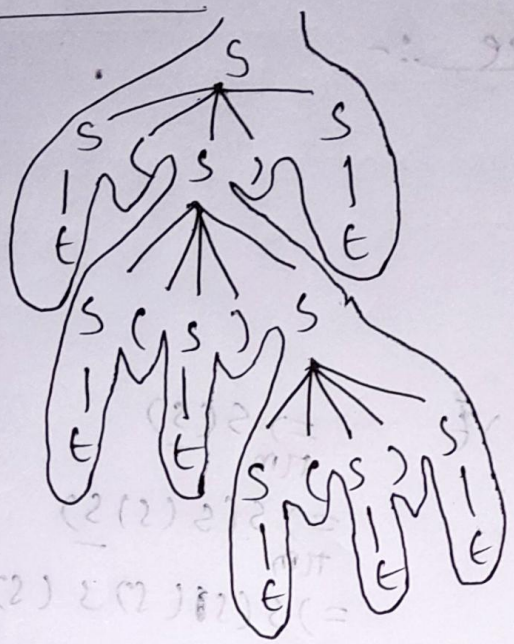
$\underset{rm}{\Rightarrow} \underline{S}(()())$

$\underset{rm}{\Rightarrow} \epsilon(()())$

$\underset{rm}{\Rightarrow} (()())$

P.T.O

So Parse tree :-



~~Str :- ε~~

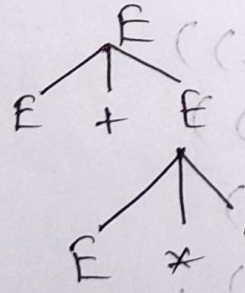
Str :- ε (ε (ε) ε (ε) ε) ε

=> (( ) ( ))

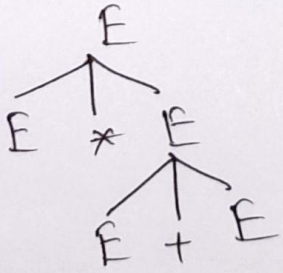
$E \rightarrow E+E$
$\quad | E*E$
$\quad | id$

id → 2 | 3 | 5

Tree - 1 :-



Tree - 2



* if any grammar generates 2 different parse tree then it is called ambigious grammar.

=) You cannot use an ambigous grammar because it confuses the compiler.

=) compiler need to be deterministic.

A grammar is called ambigous if we find :-

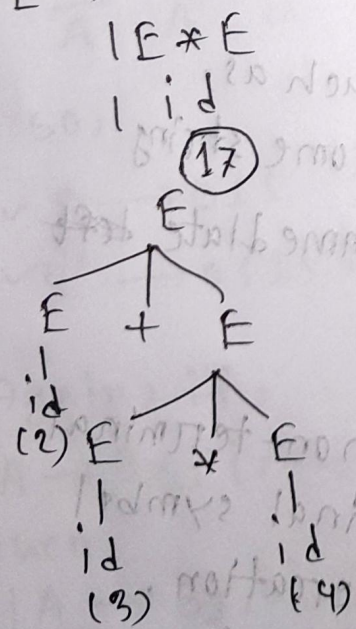
— More than one LMD or
— More than one RMD or
— More than one parse tree.

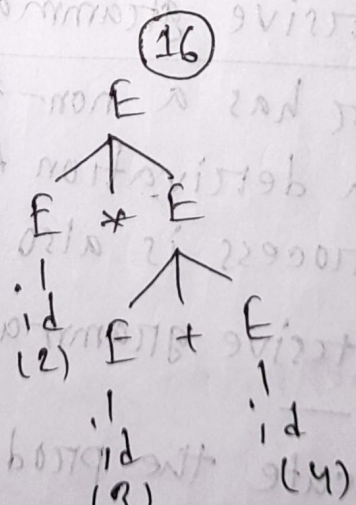
10.05.23

2nd (CC)

Left recursive grammar → Non deterministic grammar.

$E \rightarrow E+E \mid E*E \mid id$





Precedence priyority :- (+, −, *, /, %)

(setting priyority for operators)

* if there is non-terminal after terminal symbol than it is a infinite or left-recursive grammar.

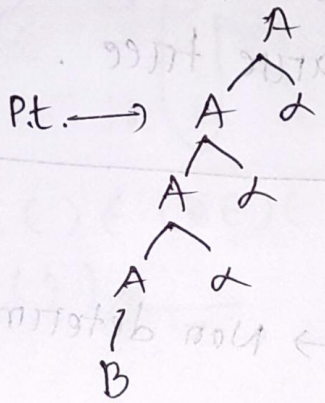
Ex! A(){

A();

α;

}

$A \rightarrow A\alpha \mid B$



R.E :- $B\alpha^*$

left recursive grammar :-

A grammar has a non-terminal A such as, there is a derivation $A \Rightarrow A\alpha$. for some string α. this process is also known as immediate ~~left~~ left recursive grammar.

Problem :-

(i) Execute the production of a non-terminal symbol without checking the terminal symbol which produce infinite loop or iteration.

(ii) As it executes the production of a non-terminal symbol, so this is the way to

ambiguous.
increase probability to make a grammar ~~ambiguous~~.

(iii) A left recursive grammar makes the iteration with $B\alpha^*$, if the grammer is,

$A \rightarrow A\alpha | B$

We have to eliminate the left recursive grammer. without changing it's iteration $B\alpha^*$. Therefore the grammar should be for.

if, $A \rightarrow A\alpha | B$ is:-

then, equ. imp

$A \rightarrow BA'$

$A' \rightarrow \alpha A' | \epsilon$

$A \rightarrow \alpha A$

Ex:-

$S \rightarrow S0S1S | 01$ (immediate)

$A \quad A \quad \alpha \quad B$

so,

$S \rightarrow 01S'$

$S' \rightarrow 0S1SS' | \epsilon$

Amo

Again, if,

$A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 | \ldots | A\alpha_m | B_1 | B_2 | B_3 | \ldots | B_n$

then,

equ imp.

$A \rightarrow B_1A' | B_2A' | B_3A' | \ldots | B_nA'$

$A' \rightarrow \alpha_1A' | \alpha_2A' | \alpha_3A' | \ldots | \alpha_mA' | \epsilon$

Ex:-

$E \rightarrow E+T \mid E*T \mid a \mid b$

$A \quad A \quad \alpha_1 \quad A \quad \alpha_2 \quad B_1 \quad B_2$

then,

$E \rightarrow aE' \mid bE'$

$E' \rightarrow +TE' \mid *TE' \mid \epsilon$

Non- immediate left recursive grammer :-

$S \rightarrow Aa \mid b$

$A \rightarrow Ac \mid Sd \mid \epsilon$

$S \Rightarrow Aa$

$\Rightarrow Sda$ .... incomplete.

Question:

1) $L \rightarrow L, S \mid S$

2) $E \rightarrow Tc \mid a$

$T \rightarrow F*T \mid Eyd \mid \epsilon$

$F \rightarrow T \mid id$

3) $A \rightarrow ABd \mid Aa \mid a$

$B \rightarrow Be \mid b$

4) $S \rightarrow A$

$A \rightarrow Ad \mid Ae \mid aB \mid ac$

$B \rightarrow bBc \mid f$

Sol :-

(I) $L \rightarrow L, S \mid S$

$\underline{A} \quad \underline{A}\,\underline{\alpha} \quad \underline{B}$

$L \rightarrow SL'$

$L' \rightarrow ,SL' \mid \epsilon$

(II) $E \rightarrow Tc \mid a$

$T \rightarrow F * T \mid Eyd \mid \epsilon$

$F \rightarrow T \mid id$

1) $E \Rightarrow Tc$

$\Rightarrow Eydc$

$E \rightarrow Eydc \mid a$

$\underline{A} \quad \underline{A}\,\underline{\alpha} \quad \underline{B}$

$\therefore E \rightarrow aE'$

$E' \rightarrow ydcE' \mid \epsilon$

2) $T \rightarrow T * T \mid Tcyd \mid a$

$\underline{A} \quad \underline{A}\,\underline{\alpha_1} \quad \underline{A}\,\underline{\alpha_2} \quad \underline{B}$

$\therefore T \rightarrow aT'$

$T' \rightarrow *TT' \mid cydT' \mid \epsilon$

(III) $A \rightarrow ABd \mid Aa \mid a$

$\underline{A} \quad \underline{A}\,\underline{\alpha_1} \quad \underline{A}\,\underline{\alpha_2} \quad \underline{B}$

$B \rightarrow Be \mid b$

$\underline{B} \quad \underline{B}\,\underline{\alpha} \quad \underline{b}$

$\therefore A \rightarrow BA' \mid bA'$

$A' \rightarrow BdA' \mid aA' \mid a\epsilon$

3rd (CC)

17.05.23

A() {
  A() ;
  α
} ≈ Ba*

$A \to BA'$
$A' \to \alpha A' \mid \epsilon$

Ex:-

$E \to E+T \mid T$
$T \to T*F \mid F$
$f \to (E) \mid id$

Sol:

$\underset{A}{E} \to \underset{A}{E} + \underset{\alpha}{T} \mid \underset{B}{T}$

$E \to TE'$
$E' \to +TE' \mid \epsilon$

again,

$\underset{A}{T} \to \underset{A}{T} * \underset{\alpha}{F} \mid \underset{B}{F}$

$T \to FT'$
$T' \to *FT' \mid \epsilon$

$X \to XSb \mid Sa \mid b$
$S \to Sb \mid Xa \mid a$
$X \to Xsb \mid b$

$\longrightarrow$

$X \to Sa \mid b$
$X \to Xaa \mid aa \mid b$

Non-determinism

Non-determinism means you have numerous option on a single symbol. consider the following grammer:-

$$A \rightarrow \alpha B_1 | \alpha B_2 | \alpha B_3$$

From this grammer you have to derive αB_3. In order to derive the grammar you need to perform backtracking. In addition, the backtrack is happened because of common prefixs and it is the concept of non-determinism.

Left Factoring :—

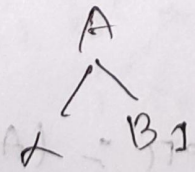
If you have a non-deterministic grammer —

$$A \rightarrow \alpha B_1 | \alpha B_2 | \alpha B_3 | \cdots | \gamma$$

then the deterministic or the determinism of the grammer could have:-

$$A \rightarrow \alpha A' | \gamma$$
$$A' \rightarrow B_1 | B_2 | B_3$$

$$A \rightarrow \alpha A'$$
$$A' \rightarrow B_1 | B_2 | B_3$$



if there is common prefix than it is non-deterministic

Ex:-

$$S \rightarrow bssaas | bssasb | bsb | a$$

Sol/

$$S \rightarrow bss' | a$$
$$S' \rightarrow saas | sasb | b$$

P.T.O →

Again, as again common prefix found.

$S' \rightarrow SaS'' \mid b$

$S'' \rightarrow aS \mid SbS \mid \epsilon$

CC (4th) 18.05.23

Elimination of (left factoring) non-deterministic does not remove the ambiguity :-

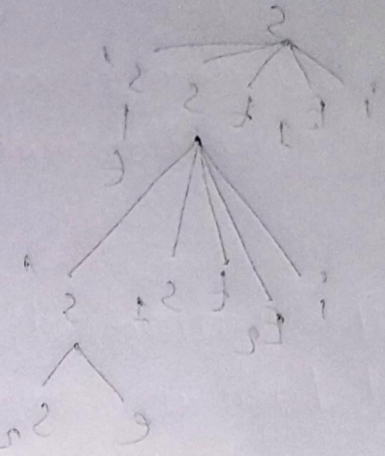
stmt → if expr then stmt
| if expr then stmt else stmt
| a

if → i
expr → E
then → t
stmt → s
else → e

S → iEtS
| iEtSeS
| a

dangling-else grammer :- (After left factoring it also can occur ambigusness)

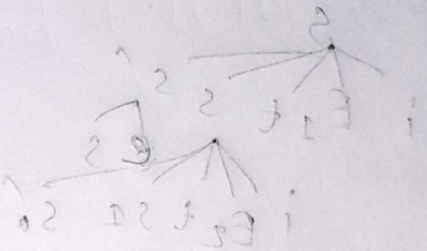
```
if () {
    if () {
    }
    else {
    }
}
```



$S \rightarrow iEtS$
$\quad | \; iEtSeS$
$\quad | \; a$

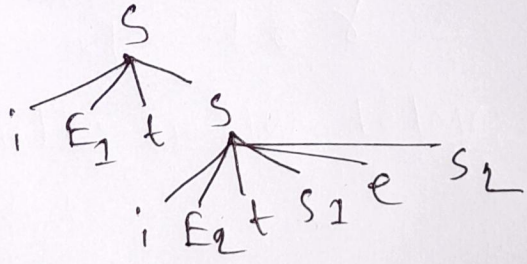
grammer :-

if E_1 then if E_2 then S_1 else S_2

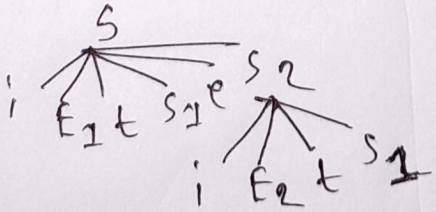


string : $iE_1tiE_2tS_1eS_2$

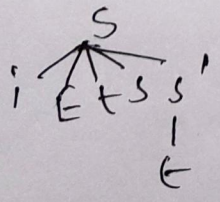
1st production :-

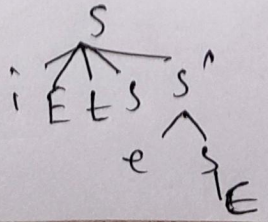


2nd production :-

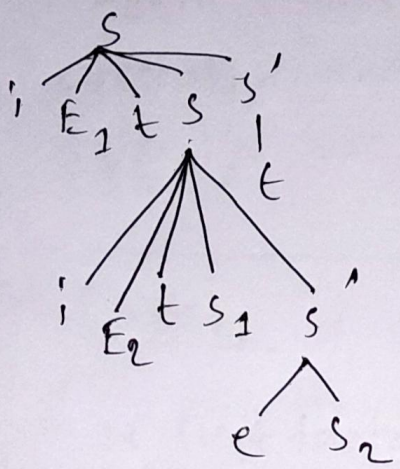


$S \rightarrow iEtSS'$
$S' \rightarrow \epsilon \; | \; eS \; | \; a$





I)



II)